TFTP Implementation

Trivial File Transfer Protocol Server

Tue, May 17, 1994

This note describes the support for the TFTP (Trivial File Transfer Protocol) for the local stations. Local stations do not have mass storage, but they have non-volatile memory that is used for the storage of local and page applications. It is therefore useful to consider this UDP-based standard protocol as a way to distribute application programs to the local stations. A motivation for doing this, rather than the present way of downloading new programs via the serial port, is that it makes it easier to access files from host machines, as modern hosts support this standard protocol. Besides, it's faster.

There are two sides to the support needed: client and server. The client side, if imple mented for the local station, would be a page application. Its implementation on a Macintosh, as recently done by Bob Peters, is an MPW tool. The client allows a user to specify a program name and a target IP address. For named programs, the name is an 8-character string, such as LOOPGRAD, that gives both the 4-character type name as well as the 4-character program name of that type. The client initiates either a read or a write transfer. The server side accepts the read / write transfer that is initiated from a client. In addition, support is included for access to the system code of a local station, or more generally, for access to an arbitrary block of memory, thus providing a kind of cheap save / restore facility.

Client logic

A read transfer copies a named program, say, from the target IP address to the initiating node, whereas a write transfer copies a named program from the initiating node to a target IP address. Blocks of 512 bytes are sent, receiving an acknowledgment for each block, until less than 512 bytes remain, when 0–511 are sent in the final block.

For a write transfer, the size of the file is known locally, so it is easy to source the file to the target IP address. Blocks are sent 512 bytes at a time, until the final one that is from 0–511 bytes in length tells the receiver (server) that it is the last. Each block is acknow ledged by the server. If no acknowledgment is received, then the block is resent. Check sums for named programs are handled by the server, as the protocol does not support them. Acknowledgments are considered enough to make the transfer itself reliable.

Server logic

The server is a local application that supports both read and write requests. A read request received by the server means that the server should transmit the program to the client. Again, the server has an easy time sourcing such requests, since the size of the program is known locally. If no acknowledgment is received, the server re-transmits the last block sent. After the final block is sent, the server activity may linger regarding that transfer to allow for the fact that the acknowledgment might not be received for the last block. Only after the acknowledgment has been received, or the server times out the transfer, should the server "close the connection."

A write request received by the server means that a program is being copied to the server node. This means that the transferred file must be read into temporary memory at first. At the conclusion of the transfer, the size is known, so the appropriate settings can be made to build the resulting program file into the non-volatile memory. The solution to this problem is to use dynamic memory to allocate each received block of up to 512 bytes. When the last block has been received, settings can be made from each allocated block of data and the block freed.

Memory access, System code "file"

A filename of the form "`MWxxxxxxxx.ssssss`" can be used to access memory data. The `xxxxxxxx` is the starting address and `ssssss` is the number of bytes of mem ory to copy. `MW` accesses the memory as words, `MB` as bytes, and `ML` as longwords. The specified address can be less than eight hexadecimal digits, and the size can be less than the six digits shown. The special name "`SYSTEM`" can be used to access the system code "file" as memory words. This name can be installed in 162bug's network boot para meters for use with the automatic network boot procedure following system reset.

Uses

Besides booting the system code via 162bug, named program access may be used to build an archive of such programs on a workstation's disk. The memory access can allow save/restore of blocks of non-volatile memory. The main use for the write sup port in the server is to allow sending system updates to a development target node during program development and debugging. The usual download page can be used to disseminate this code to other local stations, individually or via multicast addressing. Note that time-stamping of named programs is done by the server upon reception, so if a new version is only targeted to a single test node, and then disseminated from there to other nodes, the time-stamp can serve as a "version date."

Performance

The 162bug can accomplish the network boot function of its startup operation in about 2 seconds, a small part of the total of approximately 20 seconds required. The Macintosh client can transfer the system code in about 5 seconds to the TFTP server local application called LOOPTFTP. As a replacement for use of a serial port to download a new version of the system code, even at 19200 baud in 4 minutes, it's great. In addition, the S-record text file does not have to be generated. For named programs, the time is also much improved, and the S-record file generation is no longer needed.

Data request "filename" option (not implemented)

Consider support of a general data request protocol layered on top of TFTP. To keep the filename short, use only single listype / single ident requests. Also, to simplify the problem, consider only one-shot requests. The filename might have the following format: `LTxx-xxxx-xxxx.ssss`. The `xx` fields would specify the listype, the source node, and the ident in hex. The `ssss` could specify the requested number of bytes. (The fields could have a different number of hex digits than shown here.) When such a filename is received, the server would make a data request. When a reply is received, it would return the data. In case the number of bytes requested is more than 512 bytes, a temporary array will be needed to hold the data. Therefore, a limit of one such active request could be supported at one time. Note that a request for less than 512 bytes of data will result in a single reply datagram that will be acknowledged exactly once. Since a host will write a file every time such a response is received, this may not work at a high rate. But at least it can capture data using only a standard protocol.